Performance Pitfalls in the DUNE Grid Interface

Martin Nolte

October 7, 2010

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Is Performance Really Relevant?

In the scientific commumity, scalability of algorithms is most important. Why care for performance of the implementation?

- Computation easily takes days. Is waiting 10 days instead of 1 relevant?
- Computation time can be compensated by more machines. Can you spend 10 times as many computers?
- Why debug parallel code when you can still speed up the serial one?

UNI FREIBURG

Is the EntityPointer really a pointer?

The following code compiles fine but may cause segmentation faults:

```
const Entity &entity = *intersection.outside();
cout << entity.geometry().center() << endl;
```

Why?

Is the EntityPointer really a pointer?

The following code compiles fine but may cause segmentation faults:

```
const Entity &entity = *intersection.outside();
cout << entity.geometry().center() << endl;
```

Why?

Most grid implementations create entities only on demand, i.e., they exist as long as the EntityPointer exists.

⇒ Copying an EntityPointer might mean copying the entity.

Example 1: Safe but Slow

Let's have a look at the following code:

```
const Intersection &is = *intersectionIterator;
const int n = is.outside()->geometry().corners();
for( int i = 0; i < n; ++i )
{
  cout << is.outside()->geometry().corner( i )
       << endl;
}
```

In the worst case, the entity is created $n + 1$ times.

Example 1: A bit Better

Let's have a look at the following code:

```
const Intersection &is = *intersectionIterator;
const EntityPointer ep = is.outside();
const int n = ep->geometry().corners();
for( int i = 0; i < n; ++i )
  cout << ep->geometry().corner( i ) << endl;
```

Now the `EntityPointer` is created only once. Still, the entity might be reinitialized with each dereferencing.

Example 1: Better, but not Perfect

Let's have a look at the following code:

```
const Intersection &is = *intersectionIterator;
const EntityPointer ep = is.outside();
const Entity &e = *ep;
const int n = e.geometry().corners();
for( int i = 0; i < n; ++i )
  cout << e.geometry().corner( i ) << endl;
```

Now, the `EntityPointer` and `Entity` are created at most once. But what about the geometry?

Example 1: Fast but Ugly

Let's have a look at the following code:

```
const Intersection &is = *intersectionIterator;
const EntityPointer ep = is.outside();
const Entity &e = *ep;
const Geometry &geo = e.geometry();
const int n = geo.corners();
for( int i = 0; i < n; ++i )
  cout << geo.corner( i ) << endl;
```

Now, `EntityPointer`, `Entity` and `Geometry` are created at most once. We cannot do faster, but the code looks quite ugly now.

Example 1: That's It

Let's have a look at the following code:

```
const Intersection &is = *intersectionIterator;
const EntityPointer ep = is.outside();
const Geometry &geo = ep->geometry();
const int n = geo.corners();
for( int i = 0; i < n; ++i )
  cout << geo.corner( i ) << endl;
```

Now, `EntityPointer`, `Entity` and `Geometry` are still created at most once. We don't need to store the entity reference.

Avoid Using Intersection::inside

Intersection::inside creates a new EntityPointer to the inside entity.
But the intersection is obtained through code like

```
for( IIt iit = gridView.ibegin( e ); ... )
{
  EntityPointer inside = iit->inside();

  ...
}
```

In this case `inside` is just an EntityPointer to e.

GlobalIdSet vs. LocalIdSet

- Id sets are artificial DUNE structures to uniquely identify an entity.
- Grid implementations associate ids each entity, if requested ($\Rightarrow$ possible extra memory consumption).
- Global ids are unique over all processes ($\Rightarrow$ they are communicated during load balancing).
- A local id might be something available (e.g., Element$*$).

GlobalIdSet vs. LocalIdSet

- Id sets are artificial DUNE structures to uniquely identify an entity.
- Grid implementations associate ids each entity, if requested ($\Rightarrow$ possible extra memory consumption).
- Global ids are unique over all processes ($\Rightarrow$ they are communicated during load balancing).
- A local id might be something available (e.g., Element$*$).

$\Rightarrow$ Use the `GlobalIdSet` only if needed. Once requested, the grid may not delete it.

Avoiding Strict-Aliasing Warnings

Can I use `-fstrictaliasing` with DUNE?

Many of these strict aliasing warnings origin from the following situation:

```
void f ( const EntityPointer &ep );
```

Callling `f( it )` for an iterator `it` will result a strict aliasing warning in gcc 4.4 and above.

Avoiding Strict-Aliasing Warnings

Can I use `-fstrictaliasing` with DUNE?

Many of these strict aliasing warnings origin from the following situation:

```
void f ( const EntityPointer &ep );
```

Callling `f( it )` for an iterator `it` will result a strict aliasing warning in gcc 4.4 and above.

This problem can be avoided using either of following two variants of `f`:

```
template< class G, class I >
void f ( const EntityPointer< G, I > &ep );

void f ( const Entity &e );
```

In the development head of DUNE (not in the 2.0 release), such methods have been replaced by one of these variants.

UNI FREIBURG

SPGrid

SPGrid is a fast implementation of a structured, parallel grid with static load balancing (at creation time).

Features (in comparison to YaspGrid and SGrid):

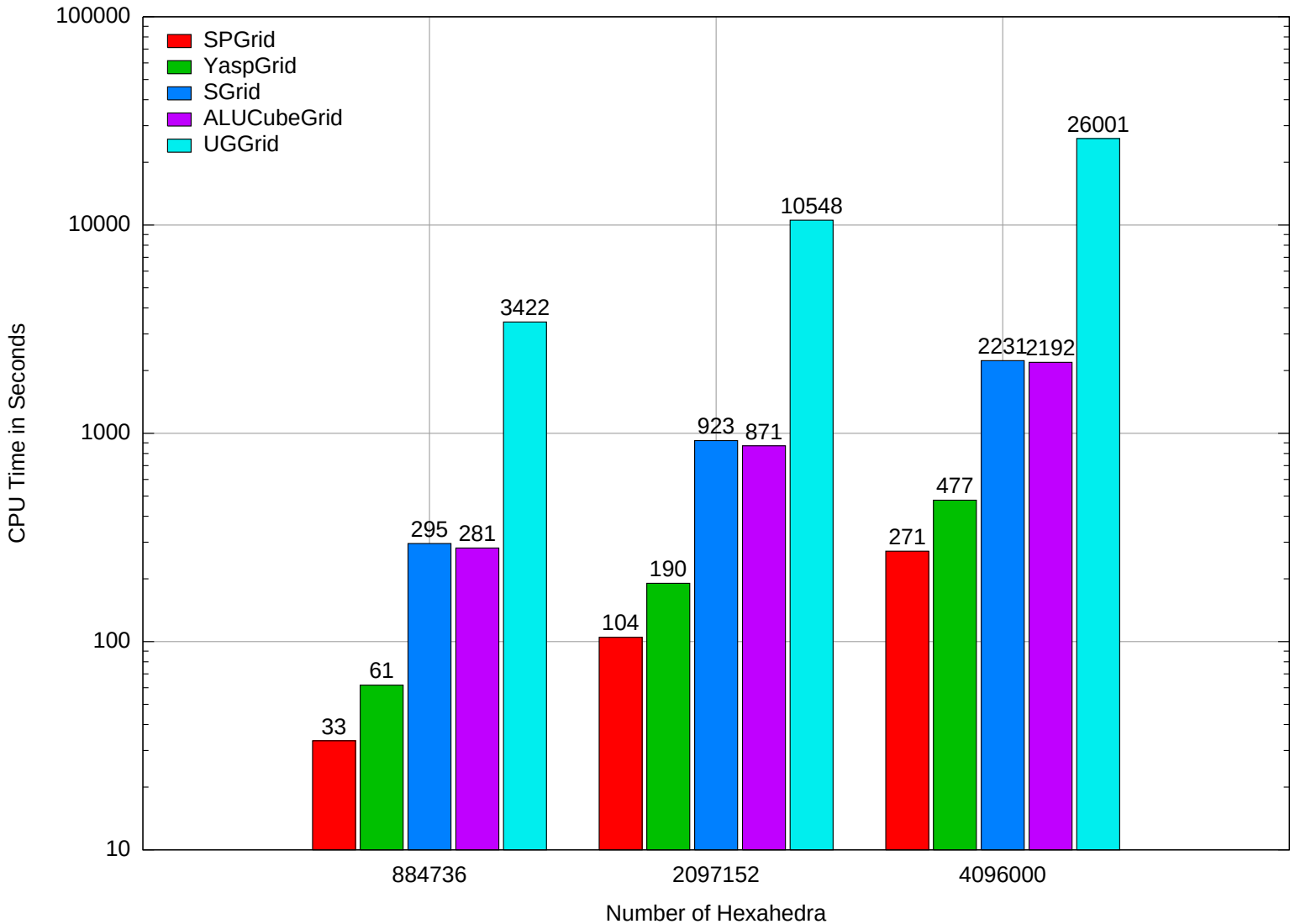
	SGrid	YaspGrid	SPGrid
supports entities of codim	$0, \ldots, d$	$0,\ d$	$0, \ldots, d$
can communicate on codim	—	$0,\ d$	$0, \ldots, d$
superentity iterators for codim	—	—	$0, \ldots, d$
supported domains	$\prod[a_i, b_i]$	$\prod[0, b_i]$	$\prod[a_i, b_i]$
supports periodicity	no	no	yes
supports anisotropic refinement	no	no	yes
supported world dimensions	$\geq d$	d	d

"SGrid is slow because it implements the complete interface."

Performance Issues of `SPGrid`

- memory consumption is independent of number of elements
- values for `jacobian`, `jacobianInverseTransposed`, `integrationElement` and `volume` stored only once for each grid level
- minimal size of on-the-fly objects like `Entity`, `Geometry`, `Intersection`, etc.
- all local geometries are stored on the grid
- data of `Entity` stored within `Geometry`

Finite Volumes on a Structured Grid



IdentityGrid's Entity

```
class IdentityGridEntity
{
  ...

  int level () const
  {
    return hostEntity_->level();
  }

  ...

  HostGridEntityPointer hostEntity_;

  ...

  const GridImp* identityGrid_;

  mutable MakeableInterfaceObject<Geometry> *geo_;
  mutable MakeableInterfaceObject<Geometry> *geoInFather_;
};
```

IdentityGrid's Iterators

```
class IdentityGridEntityPointer
{
  ...

  const GridImp* identityGrid_;

  mutable IdentityGridMakeableEntity virtualEntity_;
};
```

```
class IdentityGridLeafIterator
: public IdentityGridEntityPointer
{
  ...

  HostGridLeafIterator hostGridLeafIterator_;
  HostGridLeafIterator hostGridLeafEndIterator_;
};
```

IdGrid's Entity

```
class IdGridEntity
{
  ...

  const HostEntity *hostEntity_;
  mutable Geometry geo_;
};
```

```
class IdGridEntity
{
  ...

  const HostEntity *hostEntity_;
  mutable Geometry geo_;
};
```

IdGrid's EntityPointer

```
class IdGridEntityPointer
{
  ...

  operator const EntityPointerImp & () const
  {
    return reinterpret_cast< const EntityPointerImp & >( *this );
  }

  ...

  mutable Entity entity_;

  HostIterator hostIterator_;
};
```

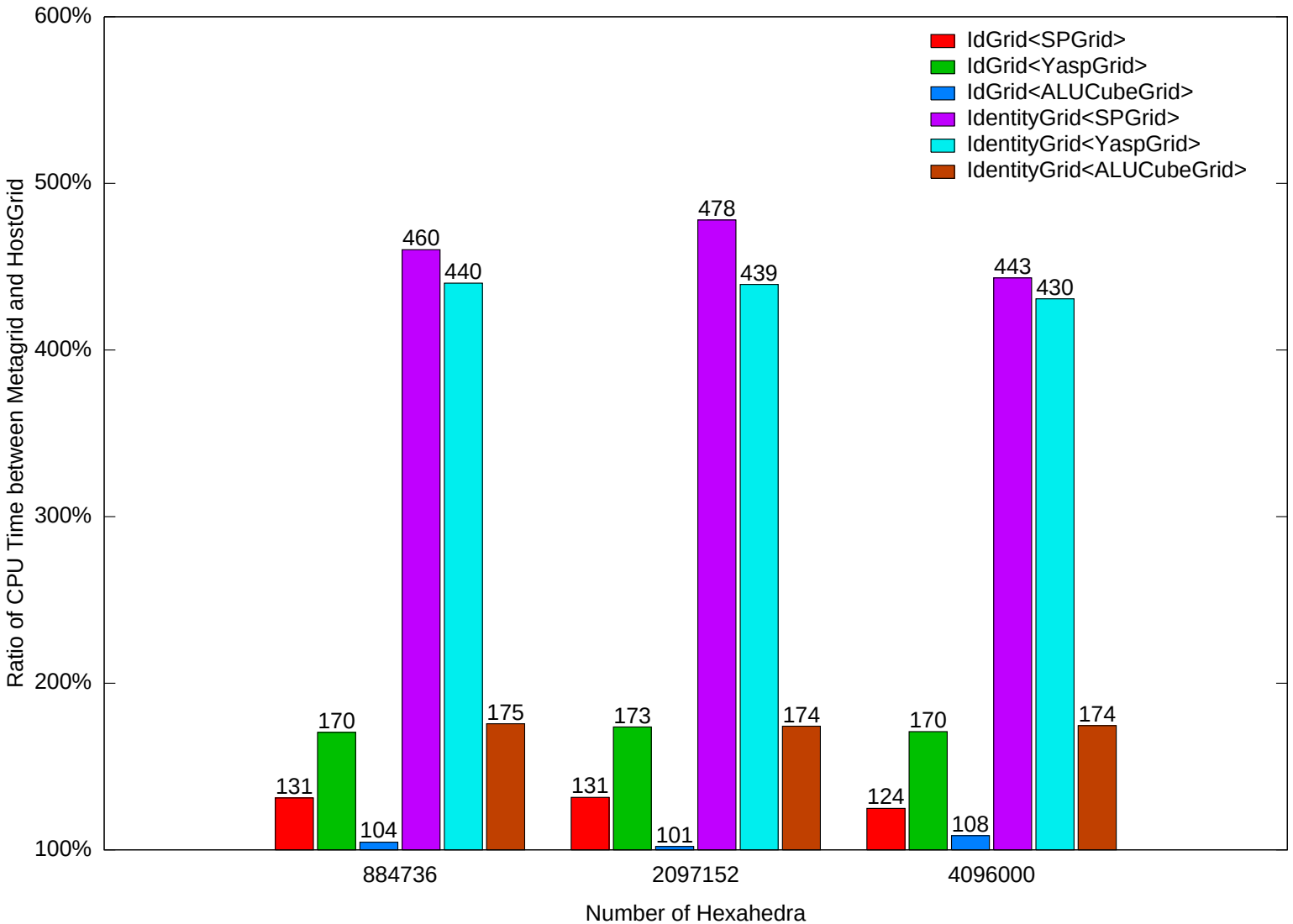
IdGrid's Iterators

```
class IdGridIterator
: public IdGridEntityPointer
{
  using IdGridEntityPointer::hostIterator_;
  using IdGridEntityPointer::releaseEntity();

  ...

  void increment ()
  {
    ++hostIterator_;
    releaseEntity();
  }
};
```

IdGrid vs. IdentityGrid: Performance



Summary

Avoiding some performance pitfalls:

- do not copy entity pointers unless required
- do not multiply obtain references like `entity()` or `geometry()`
- do not use `Intersection::inside`, you already have that entity
- use `LocalIdSet` over `GlobalIdSet`, if possible

Summary

Avoiding some performance pitfalls:

- do not copy entity pointers unless required
- do not multiply obtain references like `entity()` or `geometry()`
- do not use `Intersection::inside`, you already have that entity
- use `LocalIdSet` over `GlobalIdSet`, if possible

Other performance considerations:

- choose a grid suited for your problem
- meta grids currently add (considerable) overhead

UNI FREIBURG

Finally...

Thank you for your attention!